Abstract – Achieving quality and schedule targets continue to be serious problems in the software industry. This "software crisis" is not new, and is not helped by the fact that new Computer Science/Engineering graduates often have little experience with schedule estimation. Poor development habits acquired early on are not always corrected by the one or two Software Engineering courses commonly taught in upper level courses. We are developing ClockIt, a tool and supporting methodology that seek to improve student software development practices starting with introductory courses. As an extension or plug-in to an existing integrated development environment (IDE) ClockIt is being designed to monitor and log student development activities. It will also allow estimation of student effort by project or component, and provide reports and visualizations of student "development profiles." We believe that analysis and presentation of development profiles will provide instructors useful visual and quantitative support for teaching sound development practices. The same information will also provide students with immediate feedback with little conceptual overhead.

Index Terms – computer science education, personal software process, software engineering, software estimation.

INTRODUCTION

In 1968, a NATO conference on Software Engineering was organized to discuss what was called the "software crisis." Thirty seven years later, it was stated in a 2005 issue of Software Engineering Notes [1] that "The software crisis is still with us. In fact, it is worse than it has ever been, and we see evidence of this crisis regularly." Software systems are delivered late, exceed their budgets and contain errors. Attempts to address the problems of software quality and timeliness (or "performance" in the following) have resulted in various methodologies and tools. Among these, the Personal Software Process (PSP) [2] is a well-known approach that trains software developers to measure their own activities in order to improve performance. While PSP has proven to be effective [3], it lacks automation, requiring significant developer overhead to implement it. More recent approaches like PSP Dashboard [4] have addressed this by providing a degree of automation in the form of software tools that operate "alongside" the tools used by the developer. While reducing the overhead of PSP, they still require programmer input during the development process. A more recent approach, Hackystat [5], provides automation of the capture of development activity through the use of "sensors" installed into various development tools such as emacs, Junit, Ant and JBuilder. These sensors collect and log developer activity to a central server from which it can be analyzed and reported. Though a powerful tool, Hackystat is targeted for research, professional software development, or upper-level software engineering courses in universities [6] where a basic development vocabulary is assumed to exist. It also requires either a public server (where privacy issues can arise) or a local server installation (incurring administrative overhead) to provide any feedback to students or instructors. Finally, it does not support the notion of a "project" and thus does not provide direct means for training in schedule estimation.

OUR APPROACH

We believe that a partial solution to the "software crisis" is proper training of future developers—today's Computer Science majors—early in their careers. Unfortunately, design, development, testing and estimating skills are often only taught within a single software engineering course, generally late in the curriculum. By this time, students may have already established poor habits and that are difficult to correct. We are developing ClockIt to help train beginning and intermediate students in good development practices. One of our goals is to make the tool "invisible," requiring little or no "concept overhead." We are not attempting to add more to the typically-overburdened CS1/CS2 curriculum, rather we want to provide a way for instructors to demonstrate (visually) good development practices, and for students to observe their own practices in a concise, visual manner. Teaching good habits at this stage is intended to be almost peripheral rather than direct. The ClockIt toolset consists of an extension or "plug-in" to an existing IDE that records significant development events and provides analysis and visualization of them in various forms. As an extension, it is designed to provide all of its basic recording and reporting functions to any user of the IDE without requiring a server connection. In addition, it is designed to allow project data to be uploaded, analyzed, and accessed via the web.
Figure 1 shows the general structure of a ClockIt system. An interface layer connects ClockIt components to the host IDE. ClockIt includes components for capturing events directly from the IDE, from project file monitoring or from ClockIt operations. Examples are: Project or package open/close; compilation and its results; object creation, method invocations and their results; IDE termination; project file system changes; and ClockIt configuration and estimation events such as adding project start/end dates and estimate information. All events are time-stamped and logged as they occur, thus creating a "development profile" for every project. Development profiles can be analyzed singly or as a collection.

Many general questions arise when considering the teaching of "good development practices" to beginning students: How can we "picture" good vs. poor practice? What would a "fly on the wall" view of students writing their first, second, or tenth program look like on a timeline (in terms of design, editing, compilation, testing, session duration, session frequency)? How could we use this view to improve teaching? A couple of scenarios may provide a starting point:

Scenario 1: You (the instructor) have been teaching for five weeks and your students have completed four or five small projects. ClockIt is installed with BlueJ, but your students don't know it. As part of your lecture you present a slide showing a "typical" student development profile, explaining what that means. "Project 3 was assigned here, due over here. Notice how the developer has done no work until right here. Then we see lots of editing, lots of red (compile errors), finally a green (compile passed). Note there is no blue (execution or testing). Mmmm. Think we have a problem?" Then you go on to describe a better "profile", demonstrating how to first understand the problem, then design, then incrementally code, compile and test, iterating until the project is complete. You conclude by showing students how to observe their own development profiles using ClockIt and how to use them as a means of self-assessment, rather like a stopwatch for a runner.

Scenario 2: Having allowed students to get used to their development profiles, later in the semester you introduce the idea of estimating the time required to complete a component (class) within a project. You show how to enter and estimate with ClockIt (during lab or as homework) and tell students to go to work. At the end of lab (or when homework is due) you examine profiles against estimates (both duration and calendar time). From this you may better understand not only whether a student can meet an estimate, but whether the process of estimation affects the development profile and the final product.

**Project Status**

We are first providing ClockIt as an extension to BlueJ[7], an IDE that has had success with introductory programming courses using Java. We have developed the IDE event monitor and file monitor components of ClockIt, verifying feasibility and plan to have a basic prototype (including event logging and basic development profile visualization) available for trial during the summer of 2005. From this we plan to enhance the prototype for use during the 2005-2006 academic year and evaluate student profiles. We will collect historical data for individual students (all projects) and projects (all students) to see if the use of the tool and classroom teaching has an effect on development profiles and ultimately on student software development performance. We are applying for grant support and are looking at ClockIt extensions for other IDEs such as Eclipse.

**References**


October 19 – 22, 2005, Indianapolis, IN

35th ASEE/IEEE Frontiers in Education Conference

S1E-19